1. **基础概念**

**TCP/IP网络模型**

不同的硬件、操作系统之间的通信，所有的这一切都需要一种规则。而我们就把这种规则称为协议（protocol）。

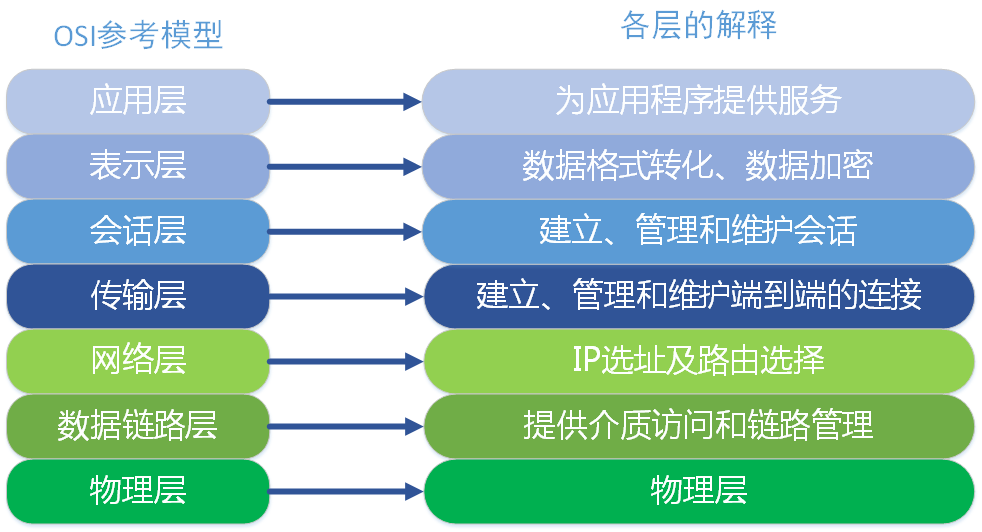
TCP/IP 是互联网相关的各类协议族的总称，比如：TCP，UDP，IP，FTP，HTTP，ICMP，SMTP 等都属于 TCP/IP 族内的协议。

**OSI模型**

它将计算机网络体系结构划分为七层,每层都可以提供**抽象良好的接口**。

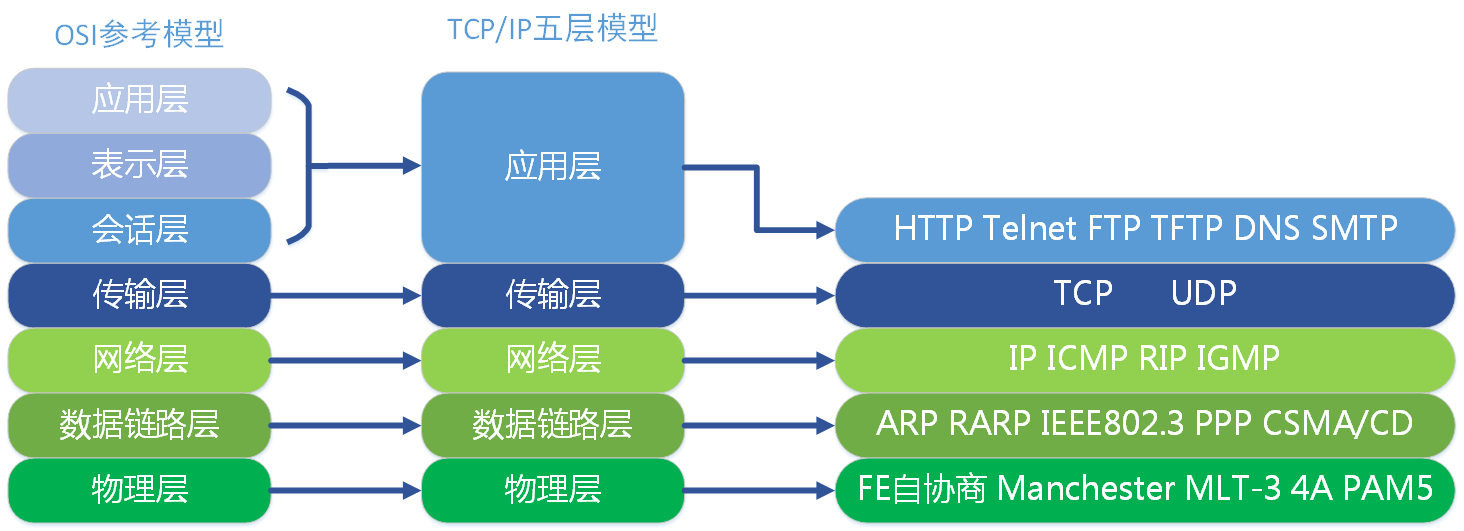
**路由器，路径选择功能：网络层**

**交换机、网桥：数据链路层**



域名系统 DNS，电子邮件的 SMTP 协议

网络层的任务就是选择合适**的网间路由和交换结点**， 确保数据及时传送。



#### IP地址的分类

**1、MAC地址**

MAC（Media Access Control，介质访问控制）地址，或称为物理地址，也叫硬件地址，用来**定义网络设备的位置，**MAC地址是网卡出厂时设定的，是固定的（但可以通过在设备管理器中或注册表等方式修改，同一网段内的MAC地址必须唯一）。**MAC地址采用十六进制数表示，长度是6个字节（48位），分为前24位和后24位。**

1、前24位叫做组织唯一标志符（Organizationally Unique Identifier，即OUI），是由IEEE的注册管理机构给不同厂家分配的代码，区分了不同的厂家。  
 2、后24位是由厂家自己分配的，称为**扩展标识符**。同一个厂家生产的网卡中MAC地址后24位是不同的。

MAC地址对应于OSI参考模型的第二层**数据链路层**，工作在数据链路层的**交换机**维护着计算机MAC地址和自身端口的数据库，交换机根据收到的数据帧中的“目的MAC地址”字段来转发数据帧。

**ARP 可以把IP映射成MAC.**

**2、IP地址**

IP地址（Internet Protocol Address），缩写为IP Adress，是一种**在Internet上的给主机统一编址的地址格式**，也称为网络协议（IP协议）地址。它为互联网上的每一个网络和每一台主机分配一个逻辑地址，常见的IP地址，**分为IPv4与IPv6两大类。**

IP地址=网络地址（前面一个）+主机地址（后面3个）

IP地址对应于OSI参考模型的第三层**网络层**，工作在**网络层的路由器根据目标IP和源IP来判断是否属于同一网段，如果是不同网段，则转发数据包。**

**ipv4到ipv6的过渡技术**

1、隧道

在IPV6网络与IPV4网络间的隧道入口处，路由器将整个IPV6数据报封装入IPV4数据报的数据字段中，IPV4分组的源地址和目的地址分别是隧道入口和出口的IPV4地址。在隧道的出口处再将IPV6**分组取出转发**给目的节点。

2、双栈

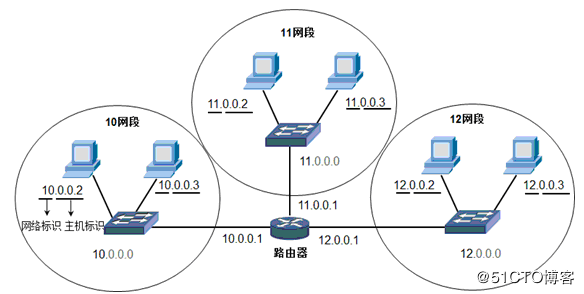
实现IPv6结点与IPv4结点互通的最直接的方式是**在IPv6结点中加入IPv4协议栈**。具有双协议栈的结点称作“IPv6/v4结点”，这些结点既可以收发IPv4分组，也可以收发IPv6分组。它们可以使用IPv4与IPv4结点互通，也可以直接使用IPv6与IPv6结点互通。双栈技术不需要构造隧道，但后文介绍的隧道技术中要用到双栈。 IPv6/v4结点可以只支持手工配置隧道，也可以既支持手工配置也支持自动隧道。

**路由器连接不同网段**，把一个IP分配给很多个主机使用，负责不同网段之间的数据转发。**交换机连接的是同一网段的计算机，端口扩展**

**路由器可以给你的局域网自动分配IP，数据传输路径选择。交换机只是用来分配网络数据的。**

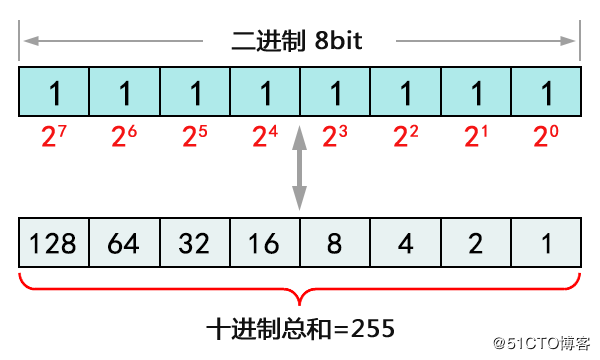
**路由器可以根据IP地址寻找下一个设备，可以处理TCP/IP协议。交换机是根据MAC地址寻址的。**

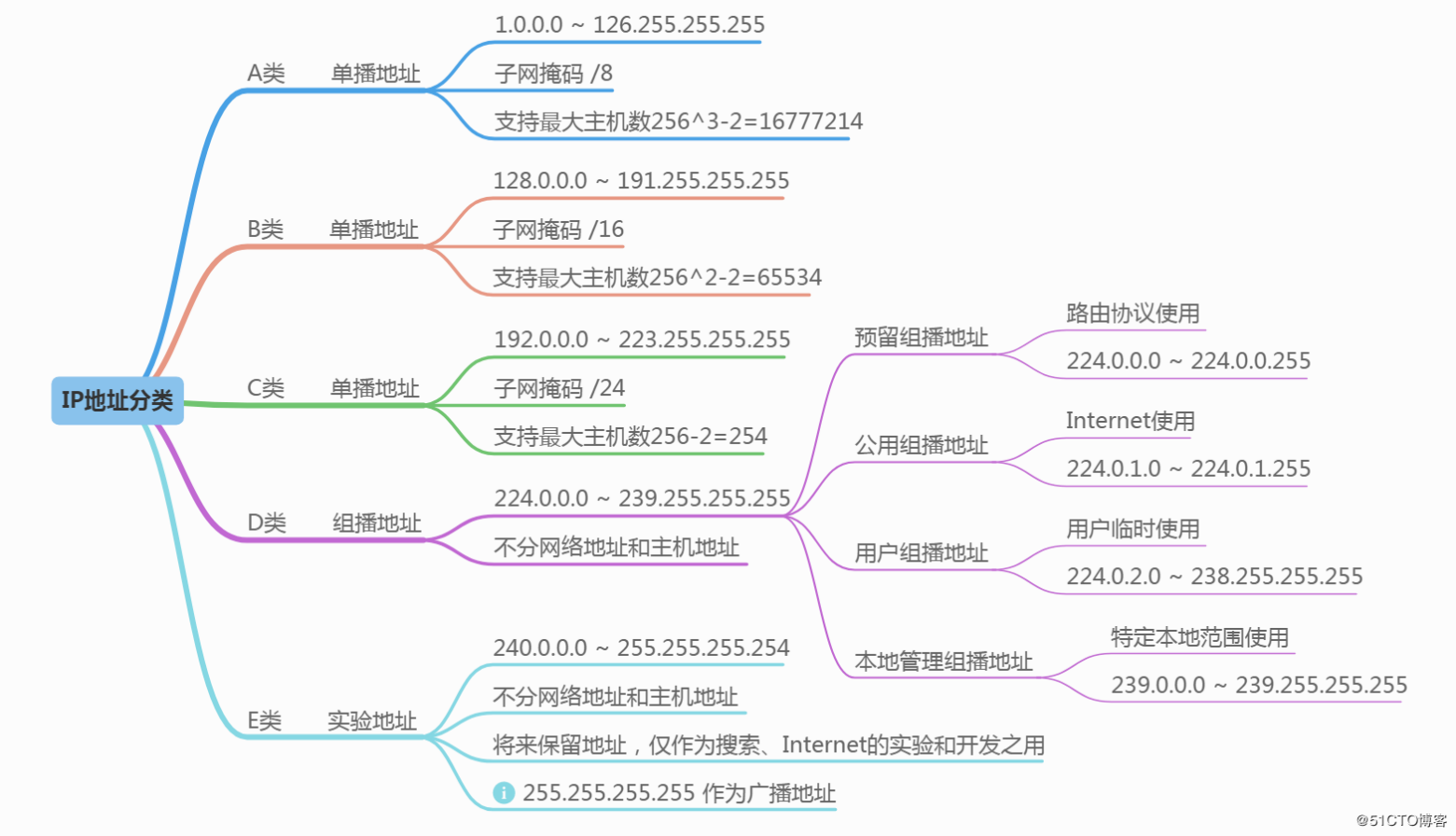
通过设置网络地址和主机地址，在互相连接的整个网络中保证每台主机的IP地址不会互相重叠，即IP地址具有了唯一性

。

在计算机二进制中，1个字节 = 8位 = 8bit（比特）

**IP地址的分配是基于网络拓扑，MAC地址的分配是基于制造商。**

个



**3.特殊ip地址**

1. 网络地址：用于表示网络本身，具有正常的网络号部分，而主机号部分全部为0的ip地址称之为网络地址，如172.16.45.0就是一个B类网络地址
2. 广播地址：用于向网络中的所有的设备进行广播。具有正常的网络号部分，而主机号部分全为1(即255)的ip地址称之为广播地址，如172.16.45.255就是一个B类的网络地址
3. **有限广播地址：指的是32位全位1(即255.255.255.255)的ip地址，用于本网广播**
4. 回送地址：网络地址不能以十进制的127作为开头，在地址中数字127保留给系统作为诊断用，称为欢送地址，如127.0.0.1用于回路测试
5. **私有地址：**只能在局域网内使用，不能在internet上使用的ip地址称为私有ip地址，私有ip地址有：
   1. **10.0.0.0～10.255.255.255，表示一个A类地址**
   2. **172.16.0.0~172.31.255.255,表示16个B类地址**
   3. **192.168.0.0～192.168.255.255，表示256个C类地址**
6. 0.0.0.0:指已经不是真正意义上的ip地址，它表示的是所有不清楚主机和目的网络，这里的不清楚指的是在本机路由表里没有特定条目指明如何到达

**4.DHCP如何实现分配IP的**； 发现阶段（DHCP客户端在网络中广播发送DHCP DISCOVER请求报文，发现DHCP服务器，请求IP地址租约）、提供阶段（DHCP服务器通过DHCP OFFER报文向DHCP客户端提供IP地址预分配）、选择阶段（DHCP客户端通过DHCP REQUEST报文确认选择第一个DHCP服务器为它提供IP地址自动分配服务）和确认阶段（被选择的DHCP服务器通过DHCP ACK报文把在DHCP OFFER报文中准备的IP地址租约给对应DHCP客户端）。

5.**子网掩码**

RFC 950定义了子网掩码的使用，子网掩码是一个32位的2进制数，其对应[网络地址](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%9C%B0%E5%9D%80)的所有位置都为1，对应于[主机地址](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA%E5%9C%B0%E5%9D%80)的所有位置都为0。

由此可知，A类网络的默认[子网掩码](https://baike.baidu.com/item/%E5%AD%90%E7%BD%91%E6%8E%A9%E7%A0%81)是255.0.0.0，B类网络的默认子网掩码是255.255.0.0，C类网络的默认子网掩码是255.255.255.0。**将子网掩码和IP地址按位进行逻辑“与”运算，**得到**IP地址的网络地址**，剩下的部分就是主机地址，从而区分出任意IP地址中的网络地址和主机地址。

子网掩码常用[点分十进制](https://baike.baidu.com/item/%E7%82%B9%E5%88%86%E5%8D%81%E8%BF%9B%E5%88%B6)表示，我们还可以用CIDR的[网络前缀](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%89%8D%E7%BC%80)法表示掩码，即“/<网络地址位数>；”。如138.96.0.0/16表示B类网络138.96.0.0的[子网掩码](https://baike.baidu.com/item/%E5%AD%90%E7%BD%91%E6%8E%A9%E7%A0%81)为255.255.0.0。

1. 子网掩码用于识别ip地址中的**网络地址和主机地址。**
2. 子网掩码也是32位二进制数字，在子网掩码中，对应于网络地址部分全用1表示，主机部分全用0表示
3. 还可以用网络前缀表示子网掩码，即"／<网络地址位数>"，如172.16.45.0/24表示C类网络172.16.45.0的子网掩码为255.255.255.0

**子网划分是通过借用ip地址的若干位主机位**

**来充当子网地址的**

**从而将原来的网络分为若干个彼此隔离的子网实现的**

 子网划分是通过借用IP地址的若干位主机位来充当子网地址从而将原网络划分为若干子网而实现的。

  划分子网时，随着子网地址借用主机位数的增多，子网的数目随之增加，而每个子网中的可用主机数逐渐减少。以C类网络为例，原有8位主机位，2的8次方即256个[主机地址](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA%E5%9C%B0%E5%9D%80)，默认[子网掩码](https://baike.baidu.com/item/%E5%AD%90%E7%BD%91%E6%8E%A9%E7%A0%81)255.255.255.0。借用1位[主机](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA)位，产生2个子网，每个子网有126个主机地址；借用2位主机位，产生4个子网，每个子网有62个主机地址……每个网中，第一个IP地址（即主机部分全部为0的IP）和最后一个IP（即主机部分全部为1的IP）不能分配给主机使用，所以每个子网的可用IP地址数为总IP地址数量减2；根据子网ID借用的主机位数，我们可以计算出划分的子网数、[掩码](https://baike.baidu.com/item/%E6%8E%A9%E7%A0%81)、每个子网主机数，列表如下：

          ① 划分子网数 ② 子网位数 ③子网掩码（二进制） ④ 子网掩码（十进制） ⑤ 每个子网主机数

          ① 1～2 ② 1 ③ 11111111.11111111.11111111.10000000 ④ 255.255.255.128 ⑤ 126

　　   ① 3～4 ② 2 ③ 11111111.11111111.11111111.11000000 ④ 255.255.255.192 ⑤ 62

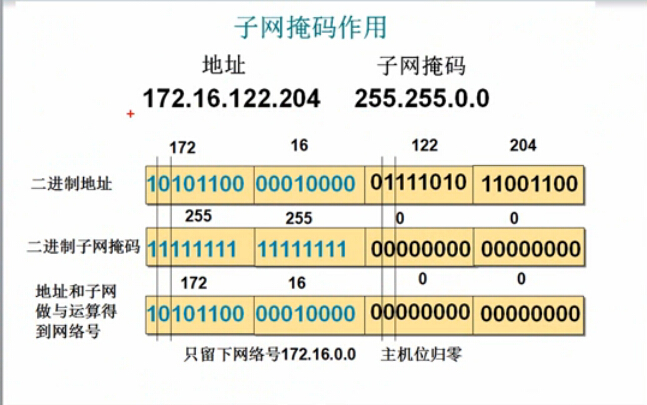
　　   ① 5～8 ② 3 ③ 11111111.11111111.11111111.11100000 ④ 255.255.255.224 ⑤ 30

　　   ① 9～16 ② 4 ③ 11111111.11111111.11111111.11110000 ④ 255.255.255.240 ⑤ 14

　　   ① 17～32 ② 5 ③ 11111111.11111111.11111111.11111000 ④ 255.255.255.248 ⑤ 6

　　   ① 33～64 ② 6 ③ 11111111.11111111.11111111.11111100 ④ 255.255.255.252 ⑤ 2

          如上表所示的C类网络中，若子网占用7位主机位时，主机位只剩一位，无论设为0还是1，都意味着主机位是全0或全1。由于主机位全0表示本网络，全1留作[广播地址](https://baike.baidu.com/item/%E5%B9%BF%E6%92%AD%E5%9C%B0%E5%9D%80)，这时子网实际没有可用[主机地址](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA%E5%9C%B0%E5%9D%80)，所以主机位至少应保留2位。



#### TCP为啥是可靠的?

TCP在传输过程中，通信双方**按照协议**进行通信；将数据截断为合理的长度；首部校验，若发现有改动则抛弃该包；通过可变大小的窗口协议进行**流量控制；**

针对乱序：在三次握手时，**序列号被初始化**，传输过程中将继续使用这个序列号，**每传送一个包，序列号进行加1；**TCP也会进行重新排序；

针对丢包：收到一个数据包过后，会用**ACK应答码进行确认**，发送方可以针对未确认的包进行重传；

针对重复：若已收到过该序列号的包，则丢弃；

#### TCP的流量控制和拥塞控制

**MSS：Maximum Segment Size，TCP一次传输发送的最大数据段长度。**

**RTT：Round-Trip Time，往返时延，表示从发送端发送数据开始，到发送端收到来自接收端的确认（接收端收到数据后便立即发送确认），总共经历的时延。**

TCP传输大块数据时，肯定需要进行**数据分段**，而每个分段所能携带的最大数据就是1个MSS，假设大块数据为100个MSS，那么发送方发送的方式大概有如下两种：

1、  每次发送1个，收到接收方确认后，才发送下1个；

2、  一口气发送100个，然后收到对方一起确认；

方式2看似很美好，实际会存在两个问题，一个是接收方的接收窗口未必能一次性接收这么多数据，另外一个是网络的带宽也不一定足够大，容易出现丢包事故。前一个问题就是标题中的**流量控制(Flow control)**，TCP采用的是**滑动窗口机制(Sliding window)，**

后一个拥塞控制(Congestion control)。

**流量控制\* TCP采用可变滑动窗口来实现流量控制。**

TCP使用端到端流量控制协议来避免发送方发送数据太快，以致TCP接收方**不能可靠地接收和处理数据**。在不同网络速度的机器进行通信的环境中，具有流量控制机制至关重要。

原理：TCP连接的两端交互作用，互相提供数据流的相关信息，包括报文段序列号、ACK号和**窗口大小**（即**接收端的可用空间**）。发送端根据这些信息**动态调节窗口大小**来控制发送，以达到流量控制的目的。

**有个标记Win，接收端告诉发送端自己还有多少缓冲区可以接收数据。**

窗口大小（即接收端的可用空间）

**总结：TCP的流量控制由滑动窗口来实现的，滑动窗口控制流量取决于接收方的窗口大小。**

**拥塞控制(Congestion control):**

* **主要想法：从丢包情况估计网络的负载情况，调整发送速率，防止过多的数据注入到网络中，这样可以使网络中的路由器或链路不致过载。**

**拥塞控制主要有四个算法：慢启动（试探，慢慢指数增长）、拥塞避免（慢启动临界值ssthresh，指数之后先行增长）、快速重传（超时重传）和快速恢复（也就是乘法减小至原来的一半，后直接开始加法增大。）。**

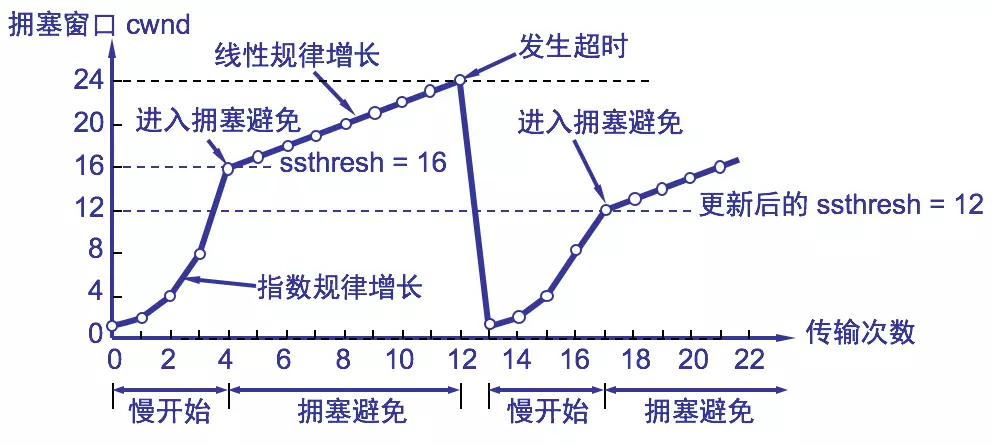
**发送方取拥塞窗口和滑动窗口的最小值作为发送上限，**即谁严格谁起决定因素。

**慢开始算法原理**

1、在主机刚刚开始发送报文段时，可先把拥塞窗口的值设置为一个最大报文段的MSS的数值。

2、在每收到一个对新的报文段的确认后，将拥塞窗口增加至多一个最大报文段MSS的数值。

3、用这种方法逐步增大发送端的拥塞窗口，这可以使分组注入到网络的**速率更加合理。**



#### TCP和UDP

是两个传输层协议,广泛应用于网络中不同主机之间传输数据。

HTTP是基于TCP，而HTTP是整个互联网的核心部分。另一个Java开发人员应该理解这两个协议的理由是, java广泛用于编写多线程、并行以及可伸缩的服务器程序。

Java还提供了丰富的基于TCP和UDP套接字编程API。

TCP代表传输控制协议和UDP代表用户数据报协议

* 基于连接vs无连接

他们之间的第一点并且最重要的区别是：TCP是面向连接的协议，而UDP是无连接的协议。这意味着当一个客户端和一个服务器通过TCP发送数据之前，必须先建立连接，他们可以通过TCP发送数据。三次握手。

UDP是无连接的协议，和点对点连接之前不需要发送消息。这就是为什么，**UDP更加适合消息的多播发布**，从单个点向多个点传输消息。

* 可靠性

TCP提供交付保证,这意味着一个使用TCP协议发送的消息是保证交付给客户端的。如果消息在传输过程中丢失,那么它将重发,这是由TCP协议本身控制的。另一方面,UDP是不可靠的,它不提供任何交付的保证。一个数据报包在运输途中可能会丢失。这就是为什么UDP是不适合保证交付的项目。

* 有序性

除了提供交付保证，为TCP也保证了消息的有序性。该消息将以从服务器端发出的同样的顺序发送到客户端，尽管这些消息到网络的另一端时可能是无序的。**TCP协议将会为你排好序。**UDP不提供任何有序性或序列性的保证。

* 速度

总而言之，TCP速度比较慢，而UDP速度比较快，因为TCP必须创建连接，以保证消息的可靠交付和有序性，他需要做比UDP多的多的事。这就是为什么UDP更适用于对速度比较敏感的应用，例如：在线视频媒体，电视广播和多人在线游戏。

* 重量级vs轻量级
* 头大小

**TCP具有比UDP更大的头。**一个TCP数据包报头的大小是**20字节，U**DP数据报报头是**8个字节。**

**TCP报头中包含序列号，ACK号，数据偏移量，保留，控制位，窗口，紧急指针，可选项，填充项，校验位，源端口和目的端口。**

**UDP报头只包含长度，源端口号，目的端口，和校验位。**

* 拥塞和流量控制

TCP有流量控制。在任何用户数据可以被发送之前，TCP需要三数据包来设置一个套接字连接。TCP处理的可靠性和拥塞控制。另一方面，**UDP不能进行流量控制。**

* 使用和应用

UDP是无状态的性质，在服务器端需要对大量客户端产生的少量请求进行应答的应用中是非常有用的。在实践中，TCP被用于金融领域，如FIX协议是一种基于TCP的协议，而UDP是大量使用在游戏和娱乐场所。

 特别需要记住的是，TCP是面向连接的，可靠的，缓慢的，可靠交付以及保证消息顺序的，而UDP是无连接的，不可靠的，没有序列保证，但是一个快速传输的协议。TCP头开销也比UDP高得多，因为它每个数据包中药**发送更多的元数据。**

#### 什么是TCP粘包半包？

1. TCP 是基于字节流的，虽然应用层和 TCP 传输层之间的数据交互是大小不等的数据块，但是 TCP 把这些数据块仅仅看成**一连串无结构的字节流，没有边界；**

2. 从 TCP 的帧结构也可以看出，在 TCP 的**首部没有表示数据长度的字段。**

基于上面两点，在使用 TCP 传输数据时，才有粘包或者拆包现象发生的可能。一个数据包中包含了发送端发送的两个数据包的信息，这种现象即为粘包。

**解决粘包半包问题**

由于底层的TCP无法理解上层的业务数据，所以在底层是**无法保证数据包不被拆分和重组**的，这个问题只能通过上层的应用协议栈设计来解决，根据业界的主流协议的解决方案，可以归纳如下。

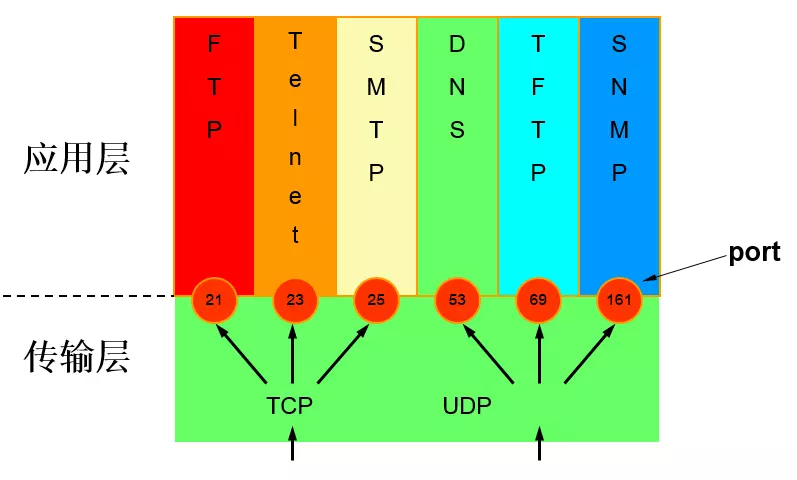
（1）**设置边界**。**在包尾增加分割符**，比如回车换行符进行分割，例如FTP协议；linebase包和delimiter包下，分别使用LineBasedFrameDecoder和DelimiterBasedFrameDecoder，如果超过规定字节长度，会报错。

（2）**消息定长，**例如每个报文的大小为固定长度200字节，如果不够，空位补空格；fixed包下，使用FixedLengthFrameDecoder

（3）**将消息分为消息头和消息体**，消息头中包含表示**消息总长度（或者消息体长度）的字段**，通常设计思路为消息头的第一个字段使用int32来表示消息的总长度，LengthFieldBasedFrameDecoder；。

如果使用 **netty 的话**，就有专门的编码器和解码器解决拆包和粘包问题了。

#### 常见应用层协议



**TCP:POP3 HTTP**

**TCP 对应的应用层协议**

FTP：定义了文件传输协议，使用 21 端口。常说某某计算机开了 FTP 服务便是启动了文件传输服务。下载文件，上传主页，都要用到 FTP 服务。

Telnet：它是一种用于远程登陆的端口，用户可以以自己的身份远程连接到计算机上，通过这种端口可以提供一种基于 DOS 模式下的通信服务。如以前的 BBS 是-纯字符界面的，支持 BBS 的服务器将 23 端口打开，对外提供服务。

SMTP：定义了简单邮件传送协议，现在很多邮件服务器都用的是这个协议，用于发送邮件。如常见的免费邮件服务中用的就是这个邮件服务端口，所以在电子邮件设置-中常看到有这么 SMTP 端口设置这个栏，服务器开放的是 25 号端口。

POP3：它是和 SMTP 对应，POP3 用于接收邮件。通常情况下，POP3 协议所用的是 110 端口。也是说，只要你有相应的使用 POP3 协议的程序（例如 Fo-xmail 或 Outlook），就可以不以 Web 方式登陆进邮箱界面，直接用邮件程序就可以收到邮件（如是163 邮箱就没有必要先进入网易网站，再进入自己的邮-箱来收信）。

HTTP：从 Web 服务器传输超文本到本地浏览器的传送协议。

**UDP 对应的应用层协议**

DNS：用于域名解析服务，将域名地址转换为 IP 地址。DNS 用的是 53 号端口。

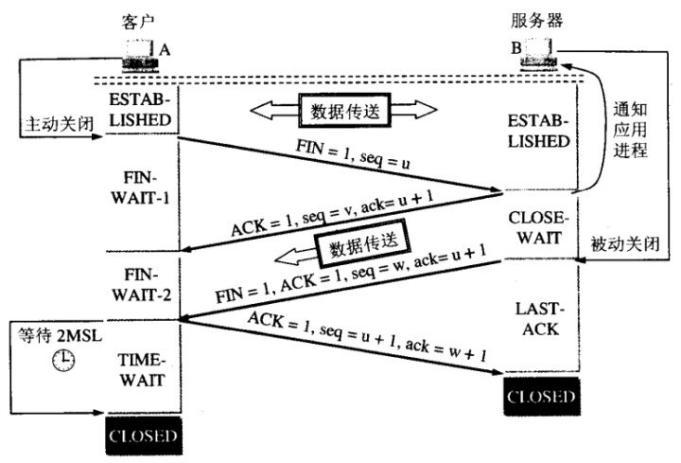
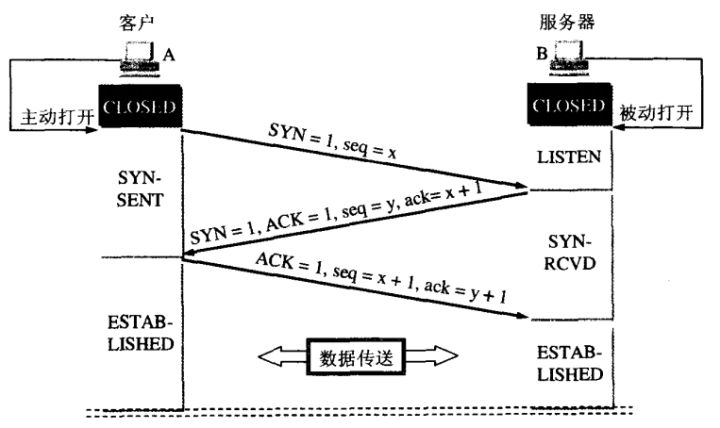
SNMP：简单网络管理协议，使用 161 号端口，是用来管理网络设备的。由于网络设备很多，无连接的服务就体现出其优势。

TFTP(Trival File Transfer Protocal)：简单文件传输协议，该协议在熟知端口 69 上使用 UDP 服务。

**不用TCP怎么连接？**

后来用telnet，curl这些tcp服务测试了一下，客户端会尝试三次SYN请求，服务器响应三次RST，然后结束链接

#### 三次握手



**TCP释放连接第二次挥手时ACK（1）,第三次挥手时ACK（1）；**

**SYN表示建立连接；确认值(Acknowledgement)，为1便是确认连接。**

**“三次握手” 的目的是为了防止已失效的链接请求报文突然又传送到了服务端，因而产生错误。**

**只要 B 发出确认，新的连接就建立了。一直等待 A 发来数据。这样，B 的很多资源就白白浪费掉了。**

**服务器陷入假性等待连接。**

**二次握手不可行**

**因为：如果由于网络不稳定，虽然客户端以前发送的连接请求以到达服务方，但服务方的同意连接的应答未能到达客户端。则客户方要重新发送连接请求，若采用二次握手，服务方收到重传的请求连接后，会以为是新的请求，就会发送同意连接报文，并新开进程提供服务，这样会造成服务方资源的无谓浪费。**

**在三次握手过程中，服务器发送SYN-ACK之后，收到客户端的ACK之前的TCP连接称为半连接(half-open connect).此时服务器处于SYN\_RECV状态.当收到ACK后，服务器转入ESTABLISHED状态.**

**客户端不断进行请求链接会怎样？DDos攻击？**

**SYN攻击步骤：**

**攻击客户端通过发包器，在短时间内伪造大量不存在的IP地址，向服务器不断地发送SYN包，服务器回复确认包SYN/ACK，并等待客户的确认，由于源地址是不存在的，服务器需要不断的重发SYN/ACK直至超时，这些伪造的SYN包将长时间占用未连接队列，正常的SYN请求被丢弃，目标系统运行缓慢，严重者引起网络堵塞甚至系统瘫痪。**

(1). DDos 攻击：

客户端向服务端发送请求链接数据包

服务端向客户端发送确认数据包

客户端不向服务端发送确认数据包，服务器一直等待来自客户端的确认

(2). DDos 预防：（没有彻底根治的办法，除非不使用TCP）

限制同时打开SYN半链接的*数目*

缩短SYN半链接的***Time out 时间***

关闭*不必要的服务*

**主要方法有SynAttackProtect保护机制、SYN cookies技术、增加最大半连接和缩短超时时间等.**

#### 为什么要四次挥手呢

TCP协议是一种面向连接的、可靠的、基于字节流的运输层通信协议。TCP是全双工模式，这就意味着，当 Client 发出FIN报文段时，只是表示 Client 已经没有数据要发送了，Client 告诉 Server，它的数据已经全部发送完毕了；但是，这个时候 Client 还是可以接受来自 Server 的数据；当 Server 返回ACK报文段时，表示它已经知道 Client 没有数据发送了，但是 Server 还是可以发送数据到 Client 的；当 Server 也发送了FIN报文段时，这个时候就表示 Server 也没有数据要发送了，就会告诉 Client ，我也没有数据要发送了，之后彼此就会愉快的中断这次TCP连接。如果要正确的理解四次分手的原理，就需要了解四次分手过程中的状态变化。

4次握手完成连接的关闭，主动关闭连接一方在第3次握手完成后发送了第四次握手的ACK包后就进入了**TIME\_WAIT状态，**必须在此状态上停留**两倍的MSL时间（两倍的数据包(2msl)最大生存时间。）**，等待2MSL时间主要目的是怕**最后一个ACK包**对方没收到，那么对方在超时后**将重发第三次握手的FIN包，**收到这个重传的 FIN+ACK 报文段。接着 A 重传一次确认，重新启动 2MSL 计时器。

若A发完确认应答后**直接进入CLOSED状态**，那么如果该**应答丢失**，B等待超时后就会**重新发送**连接释放请求，但此时A已经关闭了，**不会作出任何响应**，因此B永远无法正常关闭。

**URI和URL的区别**

URI 包含 URL 和 URN，目前 WEB 只有 URL 比较流行，所以见到的基本都是 URL。

URI（Uniform Resource Identifier，统一资源标识符

Web上可用的每种资源如HTML文档、图像、视频片段、程序等都是一个来URI来定位的

* URI一般由三部组成：

①访问资源的命名机制

②存放资源的主机名

③资源自身的名称，由路径表示，着重强调于资源。

URL（Uniform Resource Locator，统一资源定位符）

* URL是Internet上用来描述信息资源的字符串，主要用在各种WWW客户程序和服务器程序上，特别是著名的Mosaic。
* 采用URL可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。URL一般由三部组成：

①协议(或称为服务方式)

②存有该资源的主机IP地址(有时也包括端口号)

③主机资源的具体地址。如目录和文件名等

**在Java类库中，URI类不包含任何访问资源的方法，它唯一的作用就是解析。**

**相反的是，URL类可以打开一个到达资源的流。**

HTTP协议就是客户端和服务器交互的一种通迅的格式。

HTTP的诞生主要是为了能够让文档之间相互关联，形成超文本可以互相传阅

首先，我们先得知道，为什么我们要在计算机网络中分层次？？？

我们在使用的时候就可以仅仅关注我们需要关注的层次，而不用理会其他层。低耦合是一个概念。

而我们的HTTP协议是在最上层，也就是应用层

**网站通信粗略过程**

我们知道**HTTP是在应用层**中的，显然，我们在Web通信的过程中，不仅仅是需要HTTP协议的，还会涉及到其他的协议的。

**DNS：负责解析域名**

我们访问一个网页的时候，往往是通过域名来访问的www.zhongfucheng.site,而计算机通信只认的是我们的主机地址(192.168.xxx.xxx)，因此，当我们输入域名的时候，需要DNS把域名解析成主机来进行访问。

（这是因为IP地址的长度是**固定的32位，IPv6是128位**，而域名的长度并不是固定的，机器处理起来比较困难。）

**HTTP：产生请求报文数据**

对Web页面进行操作时候，就会产生HTTP报文数据，请求对应的服务端进行响应。



**TCP协议：分割HTTP数据，保证数据运输**

TCP协议采用了三次握手的方式来保证数据的准确运输，在运输的数据的时候，发送标识过去给服务器，服务器也返回标识给客户端，而客户端收到消息后再次返回标识给服务器。这样一来就保证了数据运输是可靠的。

**IP协议：**传输数据包，找到通信目的地地址。

IP协议把我们的产生的数据包发送给对方，IP地址指明了节点被分配的地址，但IP地址可能会变换，我们可以使用ARP协议来将IP地址反射为MAC地址。MAC地址是不会更改的，是网卡所属的固定地址。

在找到通信目的地之前，我们是需要不断的中转的，这过程我们称作为：“路由中转”，我们并不知道路由中转了多少次的。因此是不能全面了解到互联网中的传输状况的。

接下来就离我们比较远了，属于硬件相关的了，也就是链路层和物理层。以后复习到计算机网络的时候再来补充吧！

1. DNS解析，找到对应ip地址
2. 客户端发起http/https请求,然后交给传输层
3. 传输层将请求分成报文段，添加目标源和端口，并随机用一个本地接口封装进报头，然后交给网络层。
4. 网络层加上双方的ip地址信息，并负责路由分发。
5. 链路层中，包通过链路层发送到路由器，通过邻居协议查找给定IP地址的MAC地址，然后发送ARP请求查找目的地址，如果得到回应后就可以使用ARP的请求应答交换的IP数据包进行传输了，然后发送IP数据包到达服务器的地址。

**一次完整的HTTP请求所经历的7个步骤**

**1.DNS域名解析 2.建立TCP连接 3.发送HTTP请求 4.服务器处理请求 5.返回响应结果 6.关闭TCP连接 7.浏览器解析渲染页面**

HTTP通信机制是在一次完整的HTTP通信过程中，Web浏览器与Web服务器之间将完成下列7个步骤：

1. **建立TCP连接**

在HTTP工作开始之前，Web浏览器首先要通过网络与Web服务器建立连接，该连接是通过TCP来完成的，该协议与IP协议共同构建 Internet，即著名的TCP/IP协议族，因此Internet又被称作是TCP/IP网络。HTTP是比TCP更高层次的应用层协议，根据规则， 只有低层协议建立之后才能，才能进行更层协议的连接，因此，首先要建立TCP连接，**一般TCP连接的端口号是80。**

1. **Web浏览器向Web服务器发送请求行**

一旦建立了TCP连接，Web浏览器就会向Web服务器发送请求命令。例如：GET /sample/hello.jsp HTTP/1.1。

1. **Web浏览器发送请求头**

浏览器发送其请求命令之后，还要以头信息的形式向Web服务器发送一些别的信息，之后浏览器发送了一空白行来通知服务器，它已经结束了该头信息的发送。

1. **Web服务器应答**

客户机向服务器发出请求后，服务器会客户机回送应答， HTTP/1.1 200 OK ，应答的第一部分是协议的版本号和应答状态码。

1. **Web服务器发送应答头**

正如客户端会随同请求发送关于自身的信息一样，服务器也会随同应答向用户发送关于它自己的数据及被请求的文档。

1. **Web服务器向浏览器发送数据**

Web服务器向浏览器发送头信息后，它会发送一个空白行来表示头信息的发送到此为结束，接着，它就以Content-Type应答头信息所描述的格式发送用户所请求的实际数据。

1. **Web服务器关闭TCP连接**

**Connection:keep-alive**

一般情况下，一旦Web服务器向浏览器发送了请求数据，它就要关闭TCP连接，然后如果浏览器或者服务器在其头信息加入了这行代码：

TCP连接在发送后将**仍然保持打开状态**，于是，浏览器可以继续通过相同的连接发送请求。保持连接节省了为每个请求建立新连接所需的时间，还节约了网络带宽。

**建立TCP连接->发送请求行->发送请求头->（到达服务器）发送状态行->发送响应头->发送响应数据->断TCP连接**

#### CDN（Content Delivery Network）

**内容分发网络，是构筑在Internet上的一种先进的流量分配网络。**

**通过在现有的Internet中增加一层新的网络架构，使用户可以就近取得所需内容，提高用户访问网站的响应速度。CDN网络是在用户和服务器之间增加Cache层，如何将用户的请求引导到Cache上获得源服务器的数据，主要是通过接管DNS实现，这就是CDN的最基本的原理。实际上是通过DNS做全局负载均衡。**

**目前CDN都以缓存网站中的静态数据为主，每个CDN节点由两部分组成:负载均衡设备和高速缓存服务器。**

**用户访问某个网站的静态文件，首先要向Local DNS服务器发起请求，经过解析后返回到这个网站域名的注册服务器去解析（一般每个公司都会有一个DNS解析服务器）。这个 DNS 解析服务器通常会把请求重新CNAME 解析到另外一个域名，而这个域名最终会被指向 CDN 全局中的 DNS负载均衡服务器，再由这个GTM(智能DNS负载均衡系统) 最终分配是哪个地方的用户，返回给离这个用户最近的CDN节点。**

**负载均衡设备拥有非常好的负载均衡性能，他拥有众多的负载均衡策略（权重，动态比率，最快模式，最小连接数等），可以保证以相对较优的方式分配请求.**

**二、HTTP 方法**

客户端发送的 请求报文 第一行为请求行，包含了方法字段。

**GET获取资源**

当前网络请求中，绝大部分使用的是 GET 方法。

**POST传输实体主体**

POST 主要用来传输数据，而 GET 主要用来获取资源。

更多 POST 与 GET 的比较请见第九章。

**HEAD获取报文首部**

和 GET 方法一样，但是不返回报文实体主体部分。

主要用于确认 URL 的有效性以及资源更新的日期时间等

**PATCH对资源进行部分修改**

**PUT 也可以用于修改资源**，但是只能完全替代原始资源，PATCH 允许部分修改。

PATCH /file.txt HTTP/1.1

Host: www.example.com

Content-Type: application/example

If-Match: "e0023aa4e"

Content-Length: 100

[description of changes]

**DELETE删除文件**

与 PUT 功能相反，并且同样不带验证机制。

DELETE /file.html HTTP/1.1

**OPTIONS查询支持的方法**

查询指定的 URL 能够支持的方法。

会返回 Allow: GET, POST, HEAD, OPTIONS 这样的内容。

**HTTP提供方法的目的就是为了告知服务器该客户端想进行什么操作。**当HTTP是OPTIONS方法的时候，服务器端就会返回它支持什么HTTP方法。

当然了，**现在RESTful盛行，**也就是充分利用了HTTP协议的这些方法。

**HTTP是不保存状态的协议**

HTTP是**无状态的**，也就是说，**它是不对通信状态进行保存的。它并不知道之前通信的对方是谁**。这样设计的目的就是为了让HTTP简单化，能够快速处理大量的事务！

但是，我们经常是需要知道访问的人是谁，于是就有了Cookie技术了。

* 要是服务器端想要记住客户端是谁，那么就颁发一个cookie给客户端
* 客户端把Cookie保存在硬盘中，当下次访问服务器的时候，浏览器会自动把客户端的cookie带过去。
* 就这样，服务器就能够知道这家伙是谁了。

持久连接

在HTTP1.0的时候，每一次进行HTTP通信就会断开一次连接。如果容量很少的文本传输是没有问题的。但是如果我们访问一个网页，该网页有非常多的图片。**一个图片就算上一个HTTP请求了**。那么在中途中就不断地建立TCP连接、获取图片、断开TCP连接。

这样是非常浪费资源的，因此在HTTP1.1版本，就是持久连接了。**一次HTTP连接能够处理多个请求**。

持久连接为“管线化”方式发送成为了可能：**在一次HTTP连接里面，不需要等待服务器响应请求，就能够继续发送第二次请求**。

**提升传输效率**

在说明之前，首先我们要知道什么是实体主体

* **实体主体就是作为数据在HTTP中传输的数据**。

一般地，**实体主体可以等价为报文主体，报文主体是HTTP中的一部分**。

我们如果不使用任何手段，服务器返回的数据实体主体是原样返回的。我们可以使用两种方式来提高传输效率

* **使用压缩技术把实体主体压小，在客户端再把数据解析**
* **使用分块传输编码，将实体主体分块传输，当浏览器解析到实体主体就能够显示了。**

我们如果在下载东西的过程中断了，按照以前我们是需要重新下载的，但是现在可以在中断中继续下载。我们可以**使用到获取范围数据，这种叫做范围请求！**

这种请求只会下载资源的一部分。

* 比如我的图片下载到一半了，我们只需要下载另一半就可以组成一张完整的图片了。那么**请求的时候请求没有下载的一部分即可。**

**三、HTTP 状态码**

服务器返回的 **响应报文** 中第一行为状态行，包含了状态码以及原因短语，用来告知客户端请求的结果。

**状态码 类别 原因短语**

**1XX Informational（信息性状态码） 接收的请求正在处理**

**2XX Success（成功状态码） 请求正常处理完毕**

**3XX Redirection（重定向状态码） 需要进行附加操作以完成请求**

**4XX Client Error（客户端错误状态码） 服务器无法处理请求**

**5XX Server Error（服务器错误状态码） 服务器处理请求出错**

**1XX 信息**

100 Continue ：表明到目前为止都很正常，客户端可以继续发送请求或者忽略这个响应。

**2XX 成功**

200 OK

**204 No Content** ：请求已经成功处理，但是返回的响应报文**不包含实体的主体部分。**一般在只需要从客户端往服务器发送信息，而不需要返回数据时使用。**服务器没有新数据返回，显示页面不更新**

206 Partial Content ：表示客户端进行了**范围请求**，响应报文包含由 Content-Range 指定范围的实体内容。

**3XX 重定向**

301 Moved Permanently ：永久性请求重定向

**302 Found ：临时性重定向（sendredirect）**

303 See Other ：和 302 有着相同的功能，但是 303 **明确要求客户端应该采用 GET 方法获取资源。**

注：虽然 HTTP 协议规定 301、302 状态下重定向时不允许把 POST 方法改成 GET 方法，但是大多数浏览器都会在 301、302 和 303 状态下的重定向把 POST 方法改成 GET 方法。

304 Not Modified ：如果请求报文首部包含一些条件，例如：If-Match，If-Modified-Since，If-None-Match，If-Range，If-Unmodified-Since，如果不满足条件，则服务器会返回 304 状态码。

**307 Temporary Redirect ：临时重定向，**与 302 的含义类似，但是 307 要求浏览器不会把重定向请求的 **POST 方法改成 GET 方法。（forward）**

**4XX 客户端错误**

400 Bad Request ：**请求报文中存在语法错误。**

401 Unauthorized ：该状态码表示发送的请求需要有认证信息（BASIC 认证、DIGEST 认证）。如果之前已进行过一次请求，则表示用户认证失败。

403 Forbidden ：**请求被拒绝。**

404 Not Found

**5XX 服务器错误**

500 Internal Server Error ：**服务器正在执行请求时发生错误。**

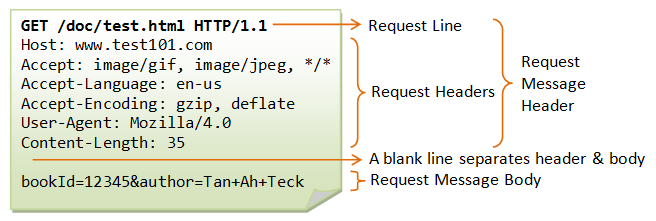
**503 Service Unavailable**：服务器暂时处于超负载或正在进行停机维护，现在无法处理请求。

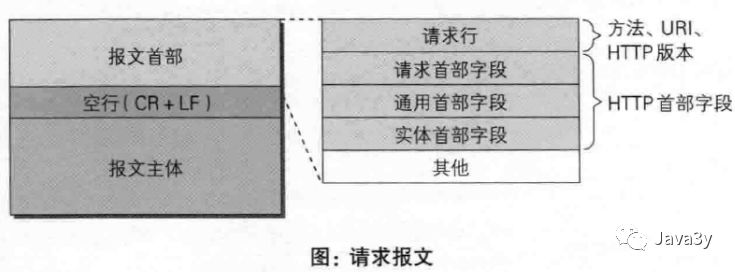
502 Bad Gateway：**作为网关或者代理工作的服务器**尝试执行请求时，从上游服务器接收到无效的响应。   
504 Gateway Time-out：作为网关或者代理工作的服务器尝试执行请求时，**未能及时**从上游服务器（URI标识出的服务器，例如HTTP、FTP、LDAP）或者辅助服务器（例如DNS）收到响应。

**四、HTTP 请求响应**

**请求和响应报文（基础）**

**1. 请求报文（手写）**

[](https://github.com/CyC2018/CS-Notes/blob/master/pics/HTTP_RequestMessageExample.png)

**HTTP请求报文**：在请求中，HTTP报文由方法、URI、HTTP版本、HTTP首部字段等部分组成。

1.请求行【描述客户端的**请求方式**、请求的资源名称，以及使用的**HTTP协议版本号**】

2**.首部字段**【描述客户端请求哪台主机，以及**客户端的一些环境信息**等】

3.一个空行

4.实体内容【**服务器向客户端回送的数据**】

**首部字段例子：**

Accept: text/html,image/\* 【浏览器告诉服务器，它支持的数据类型】

Accept-Charset: ISO-8859-1 【浏览器告诉服务器，它支持哪种**字符集**】

Accept-Encoding: gzip,compress 【浏览器告诉服务器，它支持的**压缩格式**】

Accept-Language: en-us,zh-cn 【浏览器告诉服务器，它的语言环境】

Host: www.it315.org:80【浏览器告诉服务器，它的想访问哪台主机】

If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT【浏览器告诉服务器，缓存数据的时间】

Referer: http://www.it315.org/index.jsp【浏览器告诉服务器，客户机是从那个页面来的---**反盗链**】

8.User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)【浏览器告诉服务器，浏览器的内核是什么】

Cookie【浏览器告诉服务器，**带来的Cookie是什么**】

Connection: close/Keep-Alive 【浏览器告诉服务器，请求完后是断开链接还是保持链接】

Date: Tue, 11 Jul 2000 18:23:51 GMT【浏览器告诉服务器，请求的时间】

**HTTP响应报文**

HTTP响应报文：在响应中，HTTP报文由HTTP版本、状态码（数字和原因短语）、HTTP首部字段3部分组成。

* 一个状态行【用于描述服务器对请求的处理结果。】
* 首部字段【用于描述服务器的基本信息，以及数据的描述，服务器通过这些数据的描述信息，可以通知客户端如何处理等一会儿它回送的数据】
* 一个空行
* 实体内容【服务器向客户端回送的数据】

**状态行：**

* 格式： HTTP版本号　状态码　原因叙述
* 状态行：HTTP/1.1 200 OK
* 状态码用于表示服务器对请求的处理结果，它是一个三位的十进制数。响应状态码分为5类

**首部字段例子：**

Location: http://www.it315.org/index.jsp 【服务器告诉浏览器要跳转到哪个页面】

Server:apache tomcat【服务器告诉浏览器，服务器的型号是什么】

Content-Encoding: gzip 【服务器告诉浏览器数据压缩的格式】

Content-Length: 80 【服务器告诉浏览器回送数据的长度】

Content-Language: zh-cn 【服务器告诉浏览器，服务器的语言环境】

Content-Type: text/html; charset=GB2312 【服务器告诉浏览器，回送数据的类型】

Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT【服务器告诉浏览器该资源上次更新时间】

Refresh: 1;url=http://www.it315.org【服务器告诉浏览器要定时刷新】

Content-Disposition: attachment; filename=aaa.zip【服务器告诉浏览器以下载方式打开数据】

Transfer-Encoding: chunked 【服务器告诉浏览器数据以分块方式回送】

Set-Cookie:SS=Q0=5Lb\_nQ; path=/search【服务器告诉浏览器要保存Cookie】

Expires: -1【服务器告诉浏览器不要设置缓存】

Cache-Control: no-cache 【服务器告诉浏览器不要设置缓存】

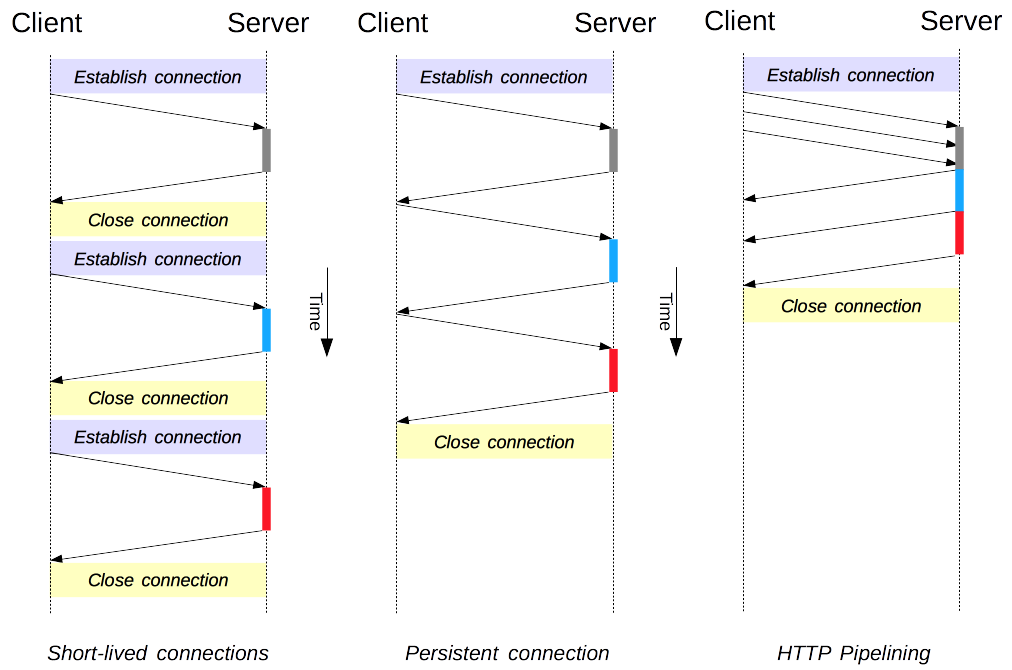
Pragma: no-cache 【服务器告诉浏览器不要设置缓存】

**Connection: close/Keep-Alive 【服务器告诉浏览器连接方式】**

Date: Tue, 11 Jul 2000 18:23:51 GMT【服务器告诉浏览器回送数据的时间】

**五、HTTP1.1**

**连接管理**

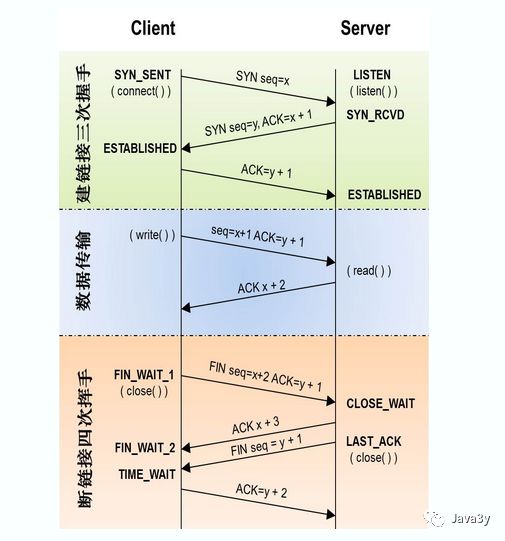
[](https://github.com/CyC2018/CS-Notes/blob/master/pics/HTTP1_x_Connections.png)

**短连接与长连接**

HTTP1.0和HTTP1.1最主要的区别就是：HTTP1.1默认是**持久化连接**！在HTTP1.0默认是**短连接**

当浏览器访问一个包含多张图片的 HTML 页面时，除了请求访问 HTML 页面资源，还会请求图片资源。如果每进行一次 HTTP 通信就要新建一个 TCP 连接，那么开销会很大。

**HTTP协议是基于TCP的，TCP每次都要经过三次握手，四次挥手，慢启动…这都需要去消耗我们非常多的资源的！**



长连接只需要建立**一次 TCP 连接就能进行多次 HTTP 通信。**

从 HTTP/1.1 开始默认是长连接的，如果要断开连接，需要由客户端或者服务器端提出断开，使用 Connection : close；

在 HTTP/1.0 之前默认是短连接的，**如果需要使用长连接，则使用 Connection : Keep-Alive。**

**数据库的连接用长连接，WEB网站的http服务一般都用短链接**

**相对于持久化连接还有另外比较重要的改动：**

* HTTP 1.1增加host字段
* HTTP 1.1中引入了Chunked transfer-coding，范围请求，实现断点续传(实际上就是利用HTTP消息头使用分块传输编码，将实体主体分块传输)
* HTTP 1.1**管线化(pipelining)理论**，客户端可以同时发出多个HTTP请求，而不用一个个等待响应之后再请求。但是回送数据给客户端的时候，客户端还是**需要按照响应的顺序来一一接收。还是会出现阻塞**
  + - 新增状态码 100
    - 新增缓存处理指令 max-age

**注意：**这个pipelining仅仅是限于理论场景下，大部分桌面浏览器仍然会选择默认关闭HTTP pipelining！

所以现在使用HTTP1.1协议的应用，都是**有可能会开多个TCP连接**的！

**范围请求**

如果网络出现中断，服务器只发送了一部分数据，范围请求可以使得客户端只请求服务器未发送的那部分数据，**从而避免服务器重新发送所有数据。**

**实际上就是利用HTTP消息头使用分块传输编码，将实体主体分块传输。**

**通信数据转发**

**1. 代理**

**代理服务器**接受客户端的请求，并且转发给其它服务器。

使用代理的主要目的是：

缓存

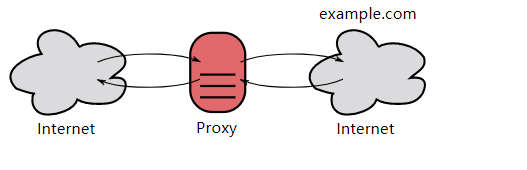
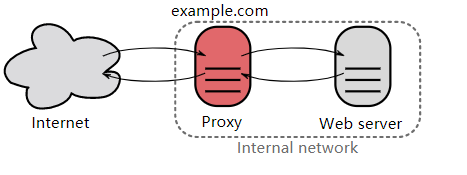
负载均衡

网络访问控制

访问日志记录

代理服务器分为正向代理和反向代理两种：

用户察觉得到正向代理的存在 而反向代理一般位于内部网络中，用户察觉不到

[](https://github.com/CyC2018/CS-Notes/blob/master/pics/a314bb79-5b18-4e63-a976-3448bffa6f1b.png)[](https://github.com/CyC2018/CS-Notes/blob/master/pics/2d09a847-b854-439c-9198-b29c65810944.png)

**2. 网关**

与代理服务器不同的是，**网关服务器会将 HTTP 转化为其它协议进行通信**，从而请求其它非 HTTP 服务器的服务。

**3. 隧道**

使用 SSL 等加密手段，在客户端和服务器之间建立一条安全的通信线路。

#### 幂等性

一次和多次请求某一个资源**对于资源本身**应该具有同样的结果。**其任意多次执行对资源本身所产生的影响均与一次执行的影响相同**。

这里需要关注几个重点：

1. 幂等不仅仅只是一次（或多次）请求对资源没有副作用（比如查询数据库操作，没有增删改，因此没有对数据库有任何影响）。
2. 幂等还包括第一次请求的时候对资源产生了副作用，但是以后的多次请求都不会再对资源产生副作用。
3. 幂等关注的是以后的多次请求是否对资源产生的副作用，**而不关注结果。**
4. 网络超时等问题，不是幂等的讨论范围。

**幂等性是系统服务对外一种承诺，外部调用失败是常态，并且失败之后必然会有重试。**

**幂等性比如：**

1. **用户在APP上连续点击了多次提交订单，后台应该只产生一个订单；**
2. **向支付宝发起支付请求，由于网络问题或系统BUG重发，支付宝应该只扣一次钱。 很显然，声明幂等的服务认为，外部调用者会存在多次调用的情况，为了防止外部多次调用对系统数据状态的发生多次改变，将服务设计成幂等性。**

**而幂等更多使用的情况是第一次请求不知道结果（比如超时）或者失败的异常情况下，发起多次请求，目的是多次确认第一次请求成功，却不会因多次请求而出现多次的状态变化。**

**为什么要设计幂等性的服务**

**幂等可以使得客户端逻辑处理变得简单，但是却以服务逻辑变得复杂为代价。满足幂等服务的需要在逻辑中至少包含两点：**

1. **首先去查询上一次的执行状态，如果没有则认为是第一次请求**
2. **在服务改变状态的业务逻辑前，保证防重复提交的逻辑**

**保证幂等策略**

**幂等需要通过唯一的业务单号来保证。也就是说相同的业务单号，认为是同一笔业务。使用这个唯一的业务单号来确保，后面多次的相同的业务单号的处理逻辑和执行效果是一致的。**

**防重复提交策略**

**上述的保证幂等方案是分成两步的，第②步依赖第①步的查询结果，无法保证原子性的。**

**在高并发下就会出现下面的情况：第二次请求在第一次请求第②步订单状态还没有修改为‘已支付状态’的情况下到来。既然得出了这个结论，余下的问题也就变得简单：把查询和变更状态操作加锁，将并行操作改为串行操作。**

**乐观锁**

**如果只是更新已有的数据，没有必要对业务进行加锁，设计表结构时使用乐观锁，一般通过version来做乐观锁，这样既能保证执行效率，又能保证幂等。例如： UPDATE tab1 SET col1=1,version=version+1 WHERE version=#version# 不过，乐观锁存在失效的情况，就是常说的ABA问题，不过如果version版本一直是自增的就不会出现ABA的情况。（从网上找了一张图片很能说明乐观锁，引用过来，出自Mybatis对乐观锁的支持）**

**防重表**

**使用订单号orderNo做为去重表的唯一索引，每次请求都根据订单号向去重表中插入一条数据。第一次请求查询订单支付状态，当然订单没有支付，进行支付操作，无论成功与否，执行完后更新订单状态为成功或失败，删除去重表中的数据。后续的订单因为表中唯一索引而插入失败，则返回操作失败，直到第一次的请求完成（成功或失败）。可以看出防重表作用是加锁的功能。**

**分布式锁**

**这里使用的防重表可以使用分布式锁代替，比如Redis。订单发起支付请求，支付系统会去Redis缓存中查询是否存在该订单号的Key，如果不存在，则向Redis增加Key为订单号。查询订单支付已经支付，如果没有则进行支付，支付完成后删除该订单号的Key。通过Redis做到了分布式锁，只有这次订单订单支付请求完成，下次请求才能进来。相比去重表，将放并发做到了缓存中，较为高效。思路相同，同一时间只能完成一次支付请求。**

**token令牌**

**这种方式分成两个阶段：申请token阶段和支付阶段。 第一阶段，在进入到提交订单页面之前，需要订单系统根据用户信息向支付系统发起一次申请token的请求，支付系统将token保存到Redis缓存中，为第二阶段支付使用。 第二阶段，订单系统拿着申请到的token发起支付请求，支付系统会检查Redis中是否存在该token，如果存在，表示第一次发起支付请求，删除缓存中token后开始支付逻辑处理；如果缓存中不存在，表示非法请求。 实际上这里的token是一个信物，支付系统根据token确认，你是你妈的孩子。不足是需要系统间交互两次，流程较上述方法复杂。**

**支付缓冲区**

**把订单的支付请求都快速地接下来，一个快速接单的缓冲管道。后续使用异步任务处理管道中的数据，过滤掉重复的待支付订单。优点是同步转异步，高吞吐。不足是不能及时地返回支付结果，需要后续监听支付结果的异步返回。**

**六、HTTPs**

**HTTP 有以下安全性问题：**

* 使用明文进行通信，内容可能会被窃听；
* 不验证通信方的身份，通信方的身份有可能遭遇伪装；
* 无法证明报文的完整性，报文有可能遭篡改。

HTTPs 并不是新协议，而是**让 HTTP 先和 SSL（Secure Sockets Layer）通信，再由 SSL 和 TCP 通信**，也就是说 HTTPs 使用了**隧道**进行通信。

**HTTPS其实是有两部分组成：HTTP + SSL / TLS**，也就是在HTTP上又加了一层处理加密信息的模块。

* 通过使用 SSL，HTTPs 具有了**加密（防窃听）、认证（防伪装）和完整性保护（防篡改）。**

**加密**

**1. 对称密钥加密**

对称密钥加密（Symmetric-Key Encryption），加密和解密使用同一密钥。

优点：运算速度快；

缺点：无法安全地将密钥传输给通信方。

**2.非对称密钥加密**

非对称密钥加密，又称公开密钥加密（Public-Key Encryption），加密和解密使用不同的密钥。

公开密钥所有人都可以获得，通信发送方获得接收方的公开密钥之后，就可以使用公开密钥进行加密，接收方收到通信内容后使用私有密钥解密。

非对称密钥除了用来加密，还可以用来进行签名。因为私有密钥无法被其他人获取，因此通信发送方使用其私有密钥进行签名，通信接收方使用发送方的公开密钥对签名进行解密，就能判断这个签名是否正确。

优点：可以更安全地将公开密钥传输给通信发送方；

缺点：运算速度慢。

**3. HTTPs 采用的加密方式**

HTTPs 采用混合的加密机制，使用**非对称密钥加密**用于传输对称密钥来保证传输过程的安全性，之后使用**对称密钥加密**进行通信来保证通信过程的效率。（下图中的 Session Key 就是对称密钥）

#### MD5和SHA加密

**MD5**

MD5算法具有以下特点：

1、压缩性：任意长度的数据，算出的MD5值长度都是固定的。

2、容易计算：从原数据计算出MD5值很容易。

3、抗修改性：对原数据进行任何改动，哪怕只修改1个字节，所得到的MD5值都有很大区别。

4、弱抗碰撞：已知原数据和其MD5值，想找到一个具有相同MD5值的数据（即伪造数据）是非常困难的。

5、强抗碰撞：想找到两个不同的数据，使它们具有相同的MD5值，是非常困难的。

**SHA**

**对于长度小于2^64位的消息，SHA1会产生一个160位的消息摘要。**

该算法的思想是接收一段明文，然后以一种不可逆的方式将它转换成一段（通常更小）密文，也可以简单的理解为取一串输入码（称为预映射或信息），并把它们转化为长度较短、位数固定的输出序列即散列值（也称为信息摘要或信息认证代码）的过程。散列函数值可以说是对明文的一种"指纹"或是"摘要"所以对散列值的数字签名就可以视为对此明文的数字签名。

**认证**

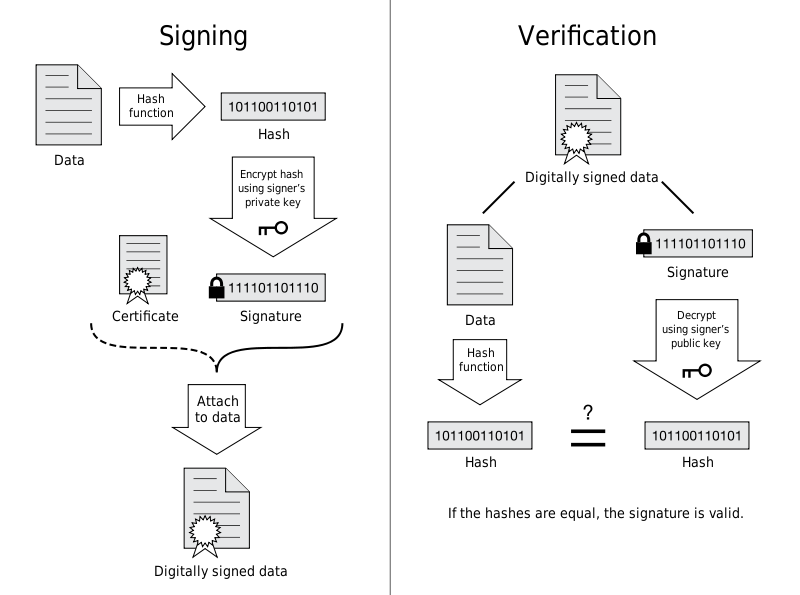
通过使用**证书**来对通信方进行认证。**真实的服务器发送的数据。**

**数字证书认证机构（CA，Certificate Authority）**是客户端与服务器双方都可信赖的第三方机构。

服务器的运营人员向 CA 提出公开密钥的申请，CA 在判明提出申请者的身份之后，会对已申请的公开密钥做数字签名，然后分配这个已签名的公开密钥，并将该公开密钥放入公开密钥证书后绑定在一起。

进行 HTTPs 通信时，服务器会把证书发送给客户端。客户端取得其中的公开密钥之后，先使用数字签名进行验证，如果验证通过，就可以开始通信了。

通信开始时，客户端需要使用服务器的公开密钥将自己的私有密钥传输给服务器，之后再进行对称密钥加密。

[](https://github.com/CyC2018/CS-Notes/blob/master/pics/2017-06-11-ca.png)

**完整性保护**

SSL 提供报文摘要功能来进行完整性保护。

HTTP 也提供了 MD5 报文摘要功能，但不是安全的。例如报文内容被篡改之后，同时重新计算 MD5 的值，通信接收方是无法意识到发生了篡改。

HTTPs 的报文摘要功能之所以安全，是因为它结合了**加密和认证**这两个操作。试想一下，加密之后的报文，遭到篡改之后，也很难重新计算报文摘要，因为无法轻易获取明文。

**通讯之路：**

远古时代：夏和女朋友聊天传输数据之间没有任何的加密，直接传输内容被看得一清二楚，毫无隐私可言

上古时期：使用对称加密的方式来保证传输的数据只有两个人知道

此时有个问题：密钥不能通过网络传输(因为没有加密之前，都是不安全的)，所以夏和女朋友先约见面一次，告诉对方密码是多少，再对话聊天。

中古时期：夏不单单要跟女朋友聊天，还要跟爸妈聊天的哇(同样不想泄漏了自己的通讯信息)。那有那么多人，难道每一次都要约来见面一次吗？(说明维护多个对称密钥是麻烦的！)--->所以用到了非对称加密

夏自己保留一份密码，独一无二的(私钥)。告诉夏女朋友，爸妈一份密码(这份密码是公开的，谁都可以拿--->公钥)。让他们给我发消息之前，先用那份我告诉他们的密码加密一下，再发送给我。我收到信息之后，用自己独一无二的私钥解密就可以了！

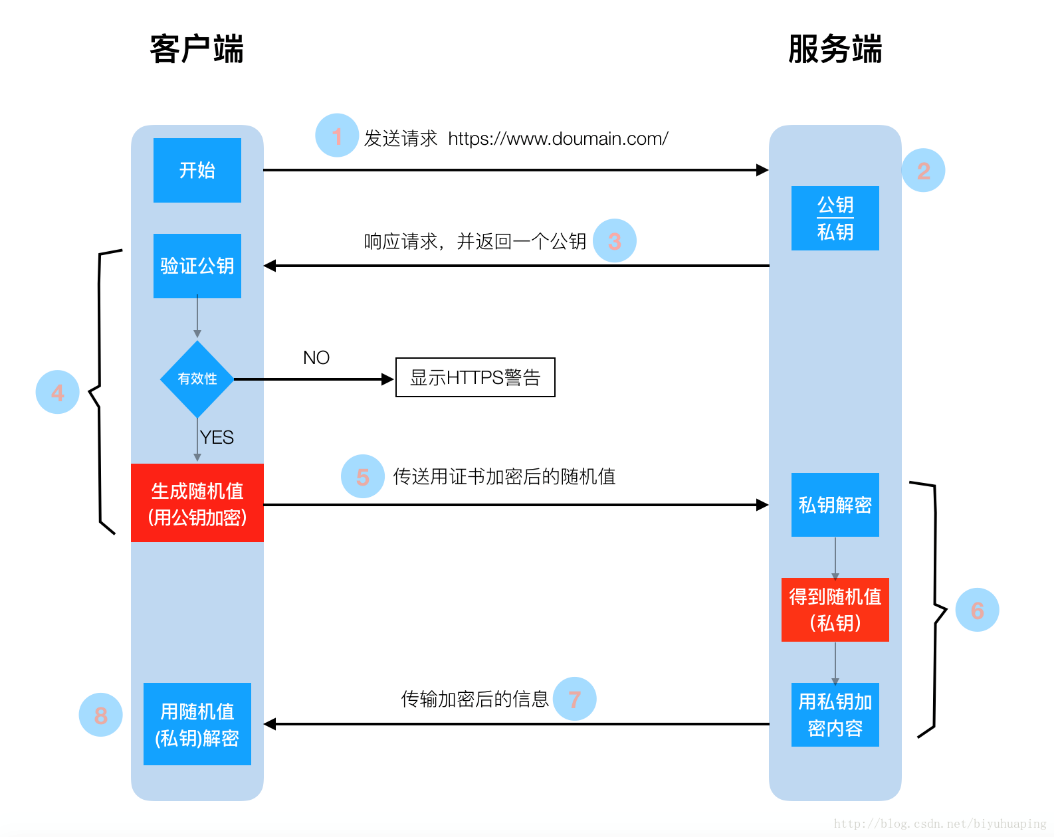
近代：此时又出现一个问题：虽然别人不知道私钥是什么，拿不到你原始传输的数据，但是可以拿到加密后的数据，他们可以改掉某部分的数据再发送给服务器，这样服务器拿到的数据就不是完整的了。

夏女朋友给夏发了一条信息”夏我喜欢你“，然后用夏给的公钥加密，发给夏了。此时不怀好意的人截取到这条加密的信息，他破解不了原信息。但是他可以修改加密后的数据再传给夏。可能夏拿到收到的数据就是”夏你今晚跪键盘吧“

现代：拿到的数据可能被篡改了，我们可以使用数字签名来解决被篡改的问题。数字签名其实也可以看做是非对称加密的手段一种，具体是这样的：得到原信息hash值，用私钥对hash值加密，另一端用公钥解密，最后比对hash值是否变了。如果变了就说明被篡改了。(一端用私钥加密，另一端用公钥解密，也确保了来源)

目前现在：好像使用了数字签名就万无一失了，其实还有问题。我们使用非对称加密的时候，是使用公钥进行加密的。如果公钥被伪造了，后面的数字签名其实就毫无意义了。讲到底：还是可能会被中间人攻击~此时我们就有了**CA认证机构来确认公钥的真实性！**

**HTTPS工作原理**



* 客户端发起HTTPS请求

输入一个https网址，然后连接到**server的443端口。**

* 服务端的配置

采用HTTPS协议的**服务器必须要有一套数字证书。**这套证书其实就是**一对公钥和私钥**。如果对公钥和私钥不太理解，可以想象成一把钥匙和一个锁箱。

* 传送证书

这个证书其实就是公钥（锁箱），只是包含了很多信息，如证书的签发者、过期时间等等。

* 客户端解析证书

客户端的TLS来完成的，**首先会验证公钥是否有效，**比如签发者、过期时间等等。证书没有问题，那么就生成一个随机值。**然后用证书对该随机值进行加密。**就好像上面说的，把随机值用锁箱锁起来。

* 传送加密信息

这部分传送的是用证书加密后的随机值，目的就是让服务端得到这个随机值，以后客户端和服务端的通信就可以**通过这个随机值来进行加密解密了。**

* 服务端解密信息

服务端用私钥解密后，得到了客户**端传过来的随机值(私钥)，**然后把内容通过该值进行对称加密。所谓对称加密就是，**将信息和私钥通过某种算法混合在一起，**这样除非知道私钥，不然无法获取内容，而正好客户端和服务端都知道这个私钥，所以只要加密算法够彪悍，私钥够复杂，数据就够安全。

* 传输加密后的信息

这部分信息是服务端用私钥加密后的信息，可以在客户端被还原。

* 客户端解密信息

客户端用之前生成的私钥解密服务端传过来的信息，于是获取了解密后的内容。整个过程第三方即使监听到了数据，也束手无策。

**七、HTTP/2.0**

HTTP2与HTTP1.1最重要的区别就是**解决了线头阻塞的问题**！其中最重要的改动是：**多路复用**

**HTTP/1.x 缺陷**

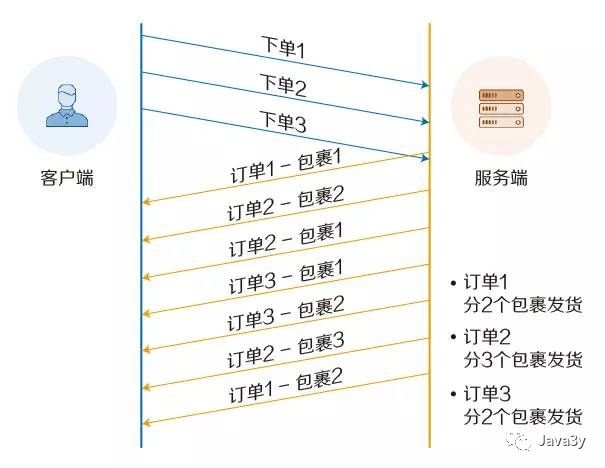
HTTP/1.x 实现简单是以牺牲性能为代价的：

客户端需要使用多个连接才能实现并发和缩短延迟；

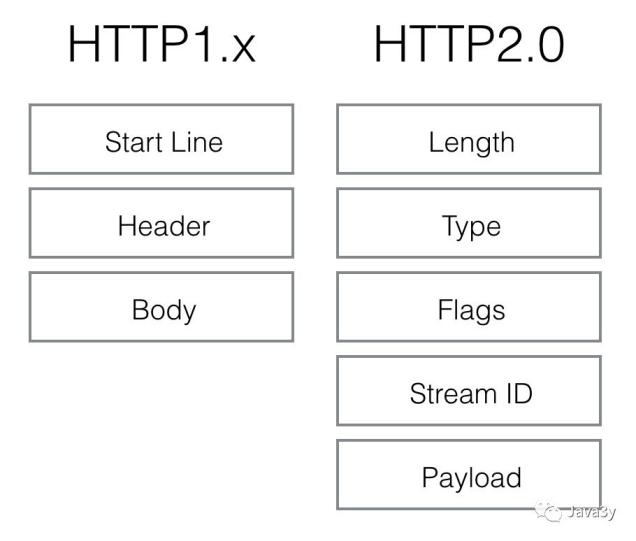
不会压缩请求和响应首部，从而导致不必要的网络流量；

不支持有效的资源优先级，致使底层 TCP 连接的利用率低下。

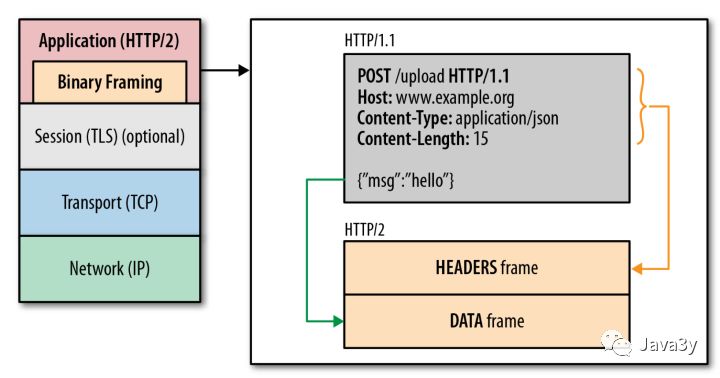
使用了HTTP2可能是这样子的：



HTTP2所有性能增强的核心在于**新的二进制分帧层**(不再以文本格式来传输了)，它定义了**如何封装http消息**并在客户端与服务器之间传输。

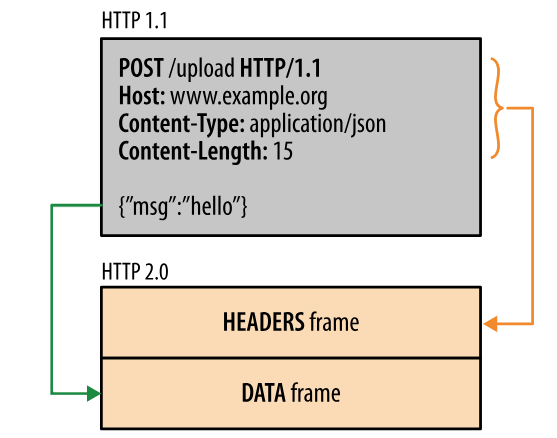


只是把原来HTTP1.x的**header和body部分用frame重新封装了一层而已**



**二进制分帧层**

**HTTP/2.0 将报文分成 HEADERS 帧和 DATA 帧，**它们都是二进制格式的。

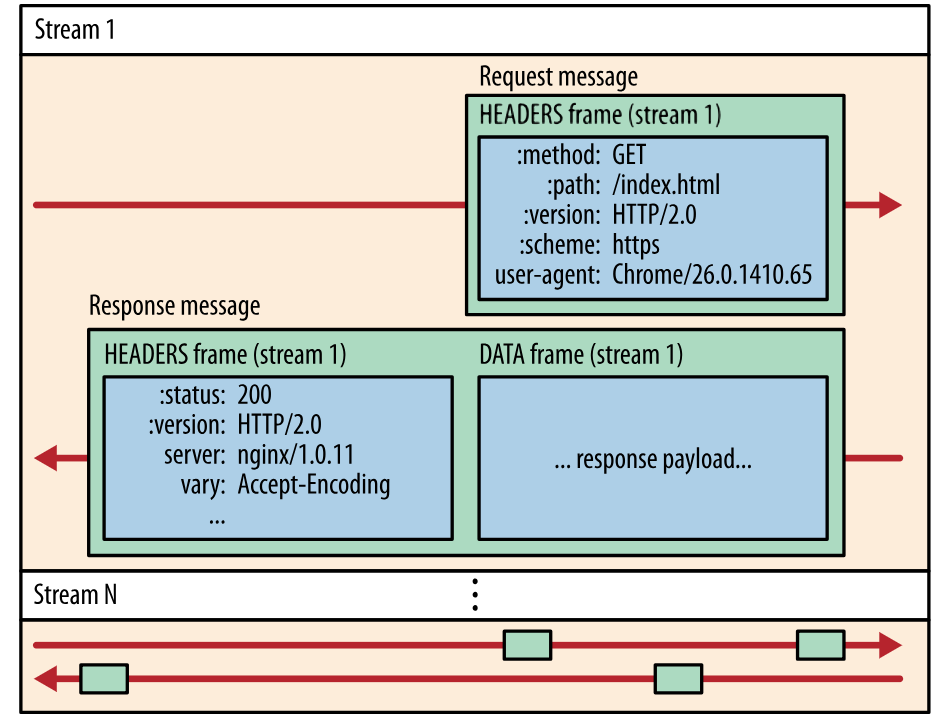
[](https://github.com/CyC2018/CS-Notes/blob/master/pics/86e6a91d-a285-447a-9345-c5484b8d0c47.png)

在通信过程中，只会**有一个 TCP 连接存在，它承载了任意数量的双向数据流（Stream）。**

一个数据流（Stream）都有一个唯一标识符和可选的优先级信息，用于承载双向信息。

消息（Message）是与逻辑请求或响应对应的完整的一系列帧。

帧（Frame）是最小的通信单位，来自不同数据流的帧可以交错发送，然后再根据每个帧头的数据流标识符重新组装。

[](https://github.com/CyC2018/CS-Notes/blob/master/pics/af198da1-2480-4043-b07f-a3b91a88b815.png)

**服务端推送**

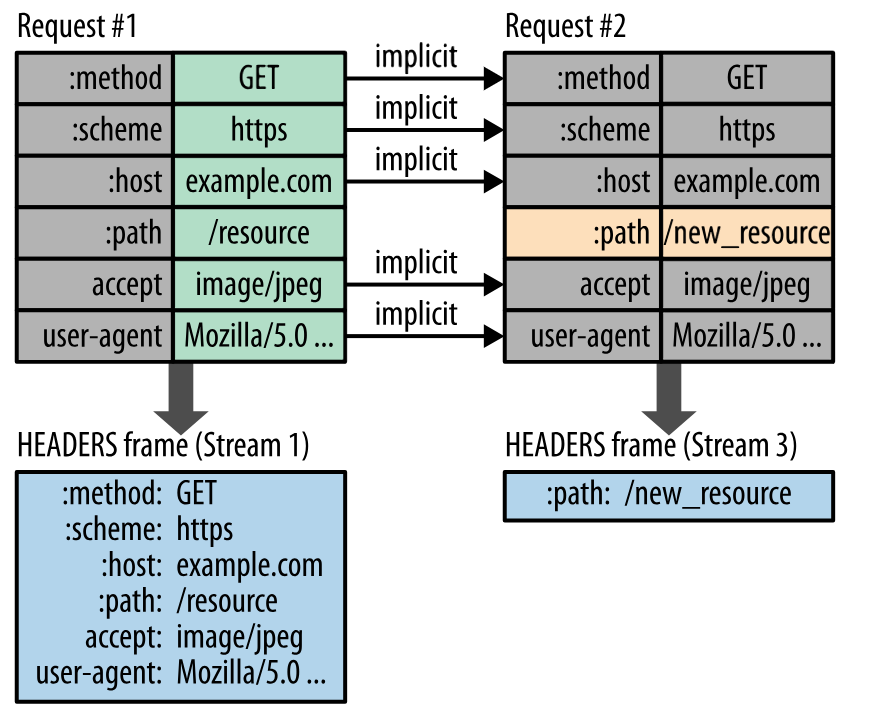
HTTP/2.0 在客户端请求一个资源时，**会把相关的资源一起发送给客户端**，客户端就不需要再次发起请求了。例如客户端请求 page.html 页面，服务端就把 script.js 和 style.css 等与之相关的资源一起发给客户端。

**首部压缩**

HTTP/1.1 的首部带有大量信息，而且每次都要重复发送。

HTTP/2.0 要求客户端和服务器同时维护和更新一个包含之前见过的首部字段表，从而避免了重复传输。

不仅如此，HTTP/2.0 也使用 Huffman 编码对首部字段进行压缩。

[](https://github.com/CyC2018/CS-Notes/blob/master/pics/_u4E0B_u8F7D.png)

HTTP2还有一些比较重要的改动：

* 使用HPACK对HTTP/2头部压缩
* 服务器推送

HTTP2推送资料：https://segmentfault.com/a/1190000015773338

* 流量控制

针对传输中的流进行控制(TCP默认的粒度是针对连接)

* 流优先级（Stream Priority）它被用来告诉对端哪个流更重要。

**九、GET 和 POST 比较**

**get和post请求区别**

1.提交形式不同：GET提交，请求的数据会附在URL之后（就是把数据放置在HTTP议头中），POST提交：把提交的数据放置在是**HTTP包的包体中**。因此，GET提交的数据会在地址栏中显示出来，而POST提交，地址栏不会改变。

2.传输数据的大小： HTTP协议并没有对传输的数据大小进行限制，HTTP协议规范也没有对URL长度进行限制。对于get请求特定浏览器和服务器对URL长度有限制；而对POST请求理论上数据不受限。但实际各个WEB服务器会规定对post提交数据大小进行限制get传送的数据量较小，**不能大于2KB**。post传送的数据量较大，一般被默认为不受限制。post基本没有限制，传不同的文件只不过要修改form里面的那个type参数

3.安全性不同：**POST的安全性要比GET的安全性高。**

4.get 和 post只是一种传递数据的方式，get也可以把数据传到服务器，他们的本质都是发送请求和接收结果。只是组织格式和数据量上面有差别

5. 对于get方式，服务器端用Request.QueryString获取变量的值，对于post方式，服务器端用Request.Form获取提交的数据。

**6.**对于GET方式的请求，浏览器会把**http header和data一并发送出去**，服务器响应200（返回数据）；

而对于POST，**浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok**（返回数据）。所以get请求会更快。

**PUT和POST使用区别**

* 1. 在HTTP中，**PUT被定义为幂等（idempotent）的方法**，POST则不是，这是一个很重要的区别。
  2. put用于向指定URL传送更新资源，是幂等的。

### 十、端口

* HTTP服务器，**默认的端口号为80/tcp**（木马Executor开放此端口）
* HTTPS（securely transferring web pages）服务器，**默认的端口号为443**/tcp 443/udp
* Telnet（不安全的文本传送），默认端口号为23/tcp（木马Tiny Telnet Server所开放的端口）
* FTP，默认的端口号为21/tcp（木马Doly Trojan、Fore、Invisible FTP、WebEx、WinCrash和Blade Runner所开放的端口）
* TFTP（Trivial File Transfer Protocol），默认的端口号为69/udp
* SSH（安全登录）、SCP（文件传输）、端口重定向，**默认的端口号为22/tcp**
* SMTP Simple Mail Transfer Protocol (E-mail），默认的端口号为25/tcp（木马Antigen、Email Password Sender、Haebu Coceda、Shtrilitz Stealth、WinPC、WinSpy都开放这个端口）
* POP3 Post Office Protocol (E-mail) ，默认的端口号为110/tcp
* WebLogic，默认的端口号为7001
* Webshpere应用程序，默认的端口号为9080
* webshpere管理工具，默认的端口号为9090
* JBOSS，默认的端口号为8080
* TOMCAT，默认的端口号为8080
* WIN2003远程登陆，默认的端口号为3389
* Symantec AV/Filter for MSE,默认端口号为 8081
* Oracle 数据库，默认的端口号为1521
* ORACLE EMCTL，默认的端口号为1158
* Oracle XDB（XML 数据库），默认的端口号为8080
* Oracle XDB FTP服务，默认的端口号为2100
* MS SQL\*SERVER数据库server，默认的端口号为1433/tcp 1433/udp
* MS SQL\*SERVER数据库monitor，默认的端口号为1434/tcp 1434/udp

**HTTP常见面试题**

**Http与Https的区别：**

* HTTP 的URL 以http:// 开头，而HTTPS 的URL 以https:// 开头
* HTTP 是不安全的，而 HTTPS 是安全的
* HTTP 标准端口是80 ，**而 HTTPS 的标准端口是443**
* 在OSI 网络模型中，HTTP工作于应用层，而HTTPS 的安全传输机制工作在传输层
* HTTP 无法加密，而HTTPS 对传输的数据进行加密
* HTTP无需证书，而**HTTPS 需要CA机构**wosign的颁发的SSL证书

**什么是Http协议无状态协议?怎么解决Http协议无状态协议?**

* 无状态协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息

也就是说，当客户端一次HTTP请求完成以后，客户端再发送一次HTTP请求，HTTP并不知道当前客户端是一个”老用户“。

* 可以使用Cookie来解决无状态的问题，Cookie就相当于一个通行证，第一次访问的时候给客户端发送一个Cookie，当客户端再次来的时候，拿着Cookie(通行证)，那么服务器就知道这个是”老用户“。

**HTTP优化方案**

我下面就简要概括一下：

* TCP复用：TCP连接复用是将多个客户端的HTTP请求复用到一个服务器端TCP连接上，而HTTP复用则是一个客户端的**多个HTTP请求通过一个TCP连接**进行处理。前者是负载均衡设备的独特功能；而后者是HTTP 1.1协议所支持的新功能，目前被大多数浏览器所支持。
* 内容缓存：将经常用到的内容进行缓存起来，那么客户端就可以直接在内存中获取相应的数据了。
* 压缩：将文本数据进行压缩，减少带宽
* SSL加速（SSL Acceleration）：使用SSL协议对HTTP协议进行加密，在通道内加密并加速
* TCP缓冲：通过采用TCP缓冲技术，可以提高服务器端响应时间和处理效率，减少由于通信链路问题给服务器造成的连接负担。

**交换机之间选择网桥ID值最小的交换机作为网络中的根网桥。**

**调用的程序在“用户态”**

**被调用的过程运行在“内核态”**

**比特填充技术：连续5个1，添加一个0。**